Facilitating Cluster Counting in Multi-dimensional Feature Space by Intermediate Information Grouping

Chloe Chun-wing Lo[1], Jishnu Chowdhury[2], Markus Hollander[3], Alexis-Walid Ahmed[3], Suraj Sood[4(✉)], Kristy Sproul[3], and Antoinette Hadgis[3]

[1] Cherrypicks Limited, Hong Kong, China
[2] Department of Computer Science, Kansas State University, Manhattan, KS, USA
[3] Sirius Project, Melbourne, USA
[4] Department of Psychology, University of West Georgia, Carrollton, GA, USA
sirusl9conf@gmail.com

Abstract. Previously, we showed that dividing 2D datasets into grid boxes could give satisfactory estimation of cluster count by detecting local maxima in data density relative to nearby grid boxes. The algorithm was robust for datasets with clusters of different sizes and distributions deviating from Gaussian distribution to a certain degree.

Given the difficulty of estimating cluster count in higher dimensional datasets by visualization, the goal was to improve the method for higher dimensions, as well as the speed of the implementation.

The improved algorithm yielded satisfactory results by looking at data density in a hypercube grid. This points towards possible approaches for addressing the curse of dimensionality. Also, a six-fold boost in average run speed of the implementation could be achieved by adopting a generalized version of quadratic binary search.

Keywords: *k*-means clustering · Unsupervised clustering · Multidimensional analysis

1 Background

Unsupervised clustering algorithms usually require the number of clusters as a parameter to yield satisfactory results. The cluster count can be easily given if the dataset is in lower dimensions where visualization can enable humans to estimate the number of clusters reasonably well.

For datasets in higher dimensions, the common practice is to run the clustering algorithm for different number of clusters, and select the number of clusters that gives the least within-cluster sum of squares (abbreviated as WSS). This approach requires human input of a range of cluster counts to test, which may not include the optimal cluster count for the best results. It is also common for multiple cluster counts to give a

D. D. Schmorrow and C. M. Fidopiastis (Eds.): HCII 2019, LNAI 11580, pp. 284–298, 2019.
https://doi.org/10.1007/978-3-030-22419-6_20

locally minimum WSS whose values are very close to each other. This ambiguity makes the selection for the ideal cluster count very difficult even with human input.

It is of great interest to develop a method that estimates the cluster count without human guess work or ambiguous results, especially for datasets in higher dimensions where it is difficult to produce a meaningful representation of cluster structure by visualization. Such a method could benefit many clustering algorithms that require an accurate cluster count input.

In the previous work, the estimation of cluster count in 2D datasets is done by dividing the 2D space of the data into grid boxes and counting the number of local maxima of number of data points found in each grid box relative to neighboring grid boxes [1]. The approach of dividing the data space into grid boxes proved to be promising in 2D space. There were two major concerns regarding this approach of estimating cluster counts: its ability to estimate cluster count accurately in higher dimensional space, and any room for speeding up the algorithm.

1.1 Extension to Higher Dimension

Applying the algorithm to higher dimension is of great interest due to the nature of human perception, which limits the ability to visualize datasets of higher dimensions accurately. While 2D datasets have almost no problems in estimating cluster count by eye from visualization, 3D data starts to suffer as the apparent number of clusters is affected by the perspective from which the data is viewed in the 3D space, as more than one clusters may appear to be one in certain angle of view if they overlap in the field of vision. As a result, cluster count estimation or cluster validation by eye is very difficult in higher dimensions, and may lead to suboptimal results in certain powerful unsupervised clustering algorithm without an accurate initial input of cluster count.

Data visualization is not the only challenge linked to increased dimensionality in machine learning. It is well known that space is dilated in higher dimensions, making the measurement of distance between data points using Euclidean distance meaningless as most data points are roughly at the same distance from each other [3]. This is termed the "curse of dimensionality". Many other definitions of distance have thus been put forward [4].

One approach that is of particular interest is the IGrid Index, which is not unlike the division of data into hypercubes [5]. It seems to suggest that the intermediate information grouping approach may work well in higher dimensional space as well.

1.2 Speed Performance

In the previous work, the attempt to estimate the number of clusters before the datasets can be passed for unsupervised clustering showed promise [1]. However, the speed of the implementation was quite slow.

The algorithm divides the whole dataset into grid boxes for 2D datasets. The best box length is the one that gives the number of boxes with single data point below a given percentage of the total data points in the dataset. In the previous version, the best box length is found by decreasing the number of portions to be divided along each axis under a geometric progression with a common ratio smaller than 1. However, this

method generates many unnecessary iterations that could be skipped by a more efficient search algorithm. Moreover, since the value progresses along the geometric decrement, it is possible that the most optimal solution is skipped, such that the final result is only second best or worse. This is a common problem with iterative machine learning solutions.

Heuristics can considerably speed up iterative approaches due to additional contextual information they utilize. It is possible to extract such contextual information during the algorithm's run to restrict the search space to a smaller range of problems that is guaranteed to contain the optimal solution. Many search algorithms employ similar approaches. For example, when an array is sorted, the order of the elements in the array can serve as a heuristic, and binary search in particular proves to be a much more efficient algorithm compared to simply searching linearly through the array.

Although the complexity of binary search is quite efficient with complexity $O(lnN)$ or more precisely $O(log_2 N)$. Quadratic binary search can further improve the time complexity to $O\left(ln\frac{N}{2}\right)$ [2].

Search algorithms like quadratic binary search often work on monotonically sorted arrays. It is possible to apply this approach to a series of values that is *almost* monotonic. In the previous version of the algorithm, the number of grid boxes with one data point generally decreased as the box length increased, with small fluctuations that did not significantly alter the overall almost monotonically decreasing trend. Quadratic binary search thus had to be adapted before it could be used to speed up this part of the algorithm.

2 Procedure

2.1 Assumptions and Definitions

We make several assumptions about the data distribution, some identical to the previous research, namely:

- Each cluster approximates a Gaussian distribution, which implies each cluster has a center where the "density" of data points peaks.
- Clusters have a low degree of overlap.
- There are more than one cluster in each dataset.
- A cluster has at least 3 data points.

The following definitions were introduced with higher dimensionality in mind:

- Density of data is defined as the number of data points per hypervolume of each hypercube.
- Hypervolume is calculated by the length of hypercube raised to the power of the dimension of the dataset.

Since all hypercubes have the same hypervolume, the identification of local maxima with highest data density can be achieved by directly identifying the hypercube with the most data points.

2.2 Algorithm

The main idea of the algorithm is to first determine the best cube length for dividing the minimum bounding box of the dataset into hypercubes of equal size and to then scan these hypercubes for local density maxima.

Optimization efforts focused on improving the search for the optimal hypercube size. Generalization to higher dimension involved changes to both optimal hypercube size selection and local maxima detection.

Different axes may have different data range, and this puts the method in danger of underestimating cluster count if certain axes have a range too small relative to other axes. Normalizing data before searching for optimal hypercube size addresses this problem.

Normalized data have very small ranges and a specialized algorithm for determining initial hypercube size is required.

Normalization. Data of each axis is normalized by transforming all of them into the standardized z-score. Since z-scores mostly range between -3 and 3, it is not sensible to use 1 as the initial hypercube length as it may be well past the optimal solution for the ideal hypercube size.

Initial Hypercube Size Determination. Let N be the total number of data points in the d-dimensional data set. The goal is to find the hypercube size that divides the minimum bounding box of the dataset into M^d d-dimensional hypercubes at the smallest possible resolution while keeping the number of empty hypercubes as small as possible, where M is the number of hypercubes along each axis.

This is accomplished by iteratively adjusting M, starting with $M_0 := \sqrt{N}$. In each iteration i, let k_i be the number of data points in the densest hypercube and compute $M_i = M_{i-1} \cdot \sqrt{k_i}$.

The square root is taken to avoid overshooting the optimal value too much. The process stops when there is no hypercube left with more than 1 data point. The final M_i is then used as a starting point in the next step.

Determination of Optimal Hypercube Size. In the previous version of the algorithm, the optimal number of hypercubes along each axis, m_{opt}, was determined by decrementing the initial M obtained in the prior step in geometric sequence and evaluating the number of data points assigned to the resulting hypercubes. To speed up this process while also avoiding to settle on a sub-optimal hypercube size, this version of the algorithm employs a modified quadratic binary search instead of geometric decrementation. Whereas quadratic binary search divides the range of potential values by 4 in each iteration, the modified version divides the range into T candidates in each iteration, whereby T is given as a parameter.

The fraction of data points alone in a hypercube, n, hereby serves as a heuristic for approximately identifying m_{opt}. This is a suitable heuristic since a grid with too small hypercubes will end up accumulating many lone data points, while a grid with too big hypercubes will overvalue noise and data points at the far edge of clusters. The algorithm takes a parameter t describing a threshold for acceptable values of n as a fraction of the volume of the dataset.

The best hypercube size is computed as follows:

1. Divide the current value range such that T candidates $m_1, \ldots, m_T$ are obtained for the determination of m_{opt}. In the first iteration, the highest possible value of m_{opt} is the initial M obtained in the previous step and the lowest possible value is 2, resulting in candidates $m_1 = M, \ldots, m_i, \ldots, m_T = 2$.
2. All m_i are rounded to integers, since the grid consists of equally sized hypercubes, and duplicate m_i are removed after rounding. If there is only one candidate, the approximated m_{opt} was found.
3. In descending order, the fraction of data points alone in a hypercube, n, is calculated for each m_i. The first m_i for which n drops below the threshold t is the lower bound for the next iteration of the search.
4. Based on the heuristic, m_{opt} lies between that m_i and the previous one, m_{i-1}, if their difference is larger than 1. In that case, steps 1 to 3 are repeated for the value range m_{i-1} to $m_i - 1$ until step 2 yields only one candidate for m_{opt}.

Cluster Count Estimation. After the value of m_{opt} is determined, the number of data points in each hypercube is compared against the nearest 8 hypercubes. The nearest hypercubes are found by selecting the hypercubes whose centre point gives the shortest Euclidean distance to the hypercube to be compared. The hypercube containing the most data points relative to its neighborhood is marked as a local maxima, and the total number of local maxima detected is returned by the algorithm as the cluster count estimation.

2.3 Time Complexity

```
# INITIAL M DETERMINATION
let N = number of data points                # assignment, O(1)
let M = sqrt(N)                              # assignment, O(1)
let d = dimension of data                    # assignment, O(1)

DO:
/* DO-WHILE LOOP:
   number of iterations for M to reach the point where
   there are no hypercube containing more than 1 data point
 */
  Map number of data points into a M^d array by dividing
    along each side of the bounding box of the data into
    M portions
                        # going through each data point, O(N)
  let max = maximum number of data points in all hypercubes
                         # going through each hypercube, O(M^d)
  if max > 1:                          # boolean checking, O(1)
    M = M*sqrt(max)                          # assignment, O(1)
  WHILE (max > 1)                      # boolean checking, O(1)
```

In theory, the number of loops required for finding the initial M depends on the maximum local density of the dataset. However, there is no way to make such a measurement except running the algorithm. This poses difficulty in determining the complexity of this part of the algorithm, especially in expressing the total number of loops and the value of M in terms of measurable metrics. Fortunately, the number of data points is finite, and this guarantees termination as the minimal distance among all data points exists when we consider a finite number of points distributed in a R^d space.

In practice, the process of initial M determination terminated within 12 loops for all test datasets. By plotting a scatter plot of the number of loops against the number of data points N, it fits well over a logarithmic curve (Fig. 1).

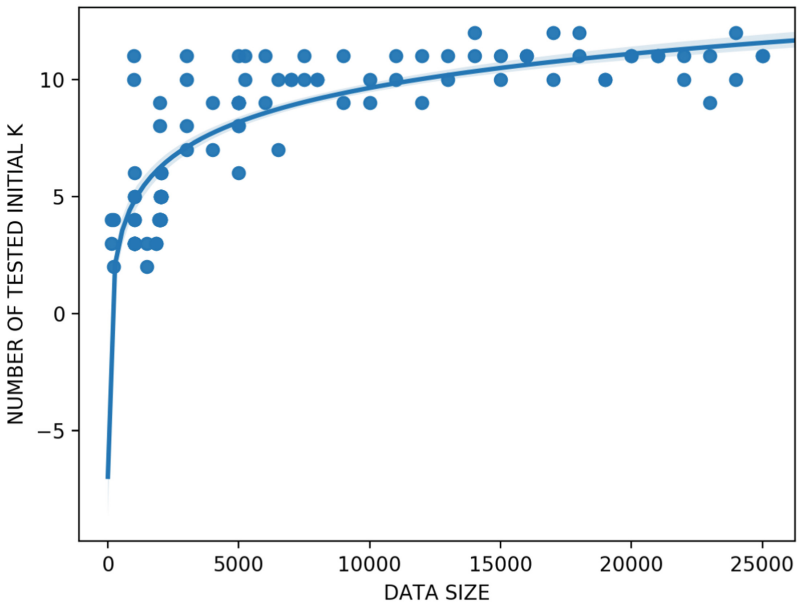

Fig. 1. The plot of number of loops against data set size (dots) is best fit by $y = 2.10158748 \cdot log(x) - 9.72704674$ (line).

The following analysis therefore works under the assumption that the number of loops is generally in $O(lnN)$. As the algorithm calculates the number of data points in each hypercube, it goes through M^d hypercubes in total. M can be at most the length of the minimum bounding box of the dataset divided by the smallest distance of the data points in one dimension, which is used as an upper bound for M in the subsequent analysis.

The big O can thus be expressed as:

$$\begin{aligned} &O\big(lnN(1+1+N+M^d+1+1+1)\big) \\ &\quad = O\big(lnN(M^d+N)\big) \end{aligned} \tag{1}$$

```
# FINDING OPTIMAL LENGTH OF HYPERCUBES
let t = tolerance level, a fraction          # assignment, O(1)
let T = number of candidate values           # assignment, O(1)
let m_max = M                                # assignment, O(1)
let m_min = 2                                # assignment, O(1)

DO:      # number of iterations for m to reach 1, O(log(2M/T))
  let r = pow(m_max/m_min, 1/(T-1))          # assignment, O(1)
  let m_candidates = []                      # assignment, O(1)
  FOR i in 1 … T:          # going through i from 1 to T, O(T)
    m_candidates.add(round(m_max/r^i))       # assignment, O(1)
  m_candidates.remove_duplicate()
                                   # go through the list, O(T)
  let idx = 1                                # assignment, O(1)

  DO:        # worse case going through all T candidates, O(T)
    let m = m_candidates[idx]                # assignment, O(1)
    Map number of data points into an m^d array by
      dividing each dimension of the dataset into m portions
                 # going through each data point,        O(N)
    Find the number of data points in each hypercube
                       # going through each hypercube, O(m^d)
    let s = number of hypercubes with one data point
                       # going through each hypercube, O(m^d)
    if s <= N*t:                        # boolean checking, O(1)
      if idx == 1:                      # boolean checking, O(1)
         let m_min = m_candidates[1]          # assignment, O(1)
         let m_max = m_candidates[1]          # assignment, O(1)
      else:
         let m_min = m_candidates[idx]        # assignment, O(1)
         let m_max = m_candidates[idx-1]-1    # assignment, O(1)
    else if idx == T:                   # boolean checking, O(1)
       let m_min = m_candidates[T]            # assignment, O(1)
       let m_max = m_candidates[T]            # assignment, O(1)

    idx = idx+1                                # assignment, O(1)

    WHILE s > N*t or idx <= T          # boolean checking, O(1)
  WHILE m_candidates.size > 1          # boolean checking, O(1)
let M = m_candidates[0]                        # assignment, O(1)
```

For an array of different m values ranging from M to 2, the quadratic binary search will give a complexity of $O\left(ln\frac{M}{2}\right)$. The division of M by 2 is a result of the division of

data into up to 4 portions in each iteration by quadratic binary search, which is a double of 2 portions in binary search. Similarly, this algorithm divides the data in up to T portions which is $\frac{T}{2}$ times of 2, and the complexity of that particular loop can be said to be $O\left(ln\frac{M}{\frac{T}{2}}\right) = O\left(ln\frac{2M}{T}\right)$. It becomes $O\left(ln\frac{M}{T}\right)$ after dropping constants. Therefore, the complexity of this part of the algorithm becomes:

$$
\begin{aligned}
&O\left(4+(2+T+T+1+T(1+N+M^d+M^d+6)+1)ln\frac{M}{T}+1\right)\\
&=O\left(4+(2+2T+1+T(1+N+2M^d+6)+1)ln\frac{M}{T}+1\right)\\
&=O\left(5+(4+2T+T(N+2M^d+7))ln\frac{M}{T}\right)\\
&=O\left((2T+T(N+2M^d))ln\frac{M}{T}\right)\\
&=O\left((2T+TN+2TM^d)ln\frac{M}{T}\right)\\
&=O\left((T+TN+TM^d)ln\frac{M}{T}\right)\\
&=O\left(T\cdot ln\frac{M}{T}(1+N+M^d)\right)\\
&=O\left(T\cdot ln\frac{M}{T}(M^d+N+1)\right)\\
&=O\left(T\cdot ln\frac{M}{T}(M^d+N)\right)
\end{aligned}
\tag{2}
$$

```
# FINDING LOCAL DENSITY MAXIMA
let maxima_centroids = []                    # assignment, O(1)
Calculate the number of data points in each hypercube
                          # going through each hypercube, O(M^d)
For each hypercube:      # going through each hypercube, O(M^d)
  calculate the distance of all other hypercubes to the current
    hypercube            # going through each hypercube, O(M^d)
  sort the hypercubes by distances to the current hypercube in
    ascending order, then by number of data points in
    descending order for the same distance
                                          #        quicksort,
O(M^2d)
  get the closest 8 hypercubes               # retrieval, O(8)
  let n_max = maximum number of data points in closest 8
    hypercubes                  # going through all boxes, O(8)
  let n_current = data points in current hypercube
```

```
                                                    #     assignment,
O(1)
    if n_current >= n_max:                # boolean checking, O(1)
      let (x_1, …, x_d) = center coordinate of current hypercube
                                                    #     assignment,
O(1)
      maxima_centroids.add((x_1, …, x_d))      # assignment, O(1)
  return maxima_centroids                 # termination of algorithm
```

The complexity of this part of the algorithm is:

$$\begin{aligned} & O(1+M^d+M^d(M^d+M^{2d}+8+8+4)) \\ & = O(1+M^d+M^d(M^d+M^{2d}+20)) \\ & = O(M^d+M^d(M^{2d})) \\ & = O(M^d+M^{3d}) \\ & = O(M^{3d}) \end{aligned} \tag{3}$$

So the overall complexity after combining Eqs. (1), (2) and (3) is:

$$O\left(lnN(M^d+N)+T\cdot ln\frac{M}{T}(M^2+N+1)+M^6\right) \tag{4}$$

Substituting $d=2$ into (5) gives:

$$O\left(lnN(M^2+N)+T\cdot ln\frac{M}{T}(M^2+N)+M^6\right) \tag{5}$$

However, we used the previous algorithm for local maxima estimation for 2D datasets for faster speed. The time complexity for local maxima estimation was M^2, so the time complexity of this version on 2D datasets should be:

$$O\left(lnN(M^2+N)+T\cdot ln\frac{M}{T}(M^2+N)+M^2\right) \tag{6}$$

The time complexity of the previous version of the algorithm for 2D datasets was:

$$\begin{aligned} & O\left(\frac{lnM}{-lnr}[N+M^2+2]+12M^2+4N+9\right) \\ & = O\left(\frac{lnM}{-lnr}[N+M^2]+M^2+N\right) \end{aligned} \tag{7}$$

The parameter T and parameter r are not shared by both expressions of the time complexity for the two versions of the algorithm. It is difficult to directly compare their performance. However, since they are parameters that are unlikely to be changed radically from dataset to dataset, we may treat them as constants. The expressions for the new version of the algorithm can be simplified to:

$$O\left(lnN(M^2+N)+lnM(M^2+N)+M^2\right) \tag{8}$$

And the one for the previous version of the algorithm can be simplified to:

$$O\left(lnM[M^2+N]+M^2+N\right) \tag{9}$$

The time complexity indicates that the new version may perform slower than the old version, but actual run time may tell a different story as the parameter T and r interacts with the algorithm in ways that is difficult to compare their effect on both versions of the algorithm.

2.4 Parameters

We for comparison, the tolerance level t, describing the acceptable fraction of data points alone in a hypercube, was set to 0.004 and the number of candidate values T to 20 for all datasets.

2.5 Test Data

We tested our algorithm on data sets from the University of Eastern Finland [6] and Kaggle [7, 8].

2.6 Hardware and Software

The algorithm was run on a MacBook Pro (Retina, 13-inch, late 2013), 2.4 GHz Intel Core Duo i5, 8 GB 1600 MHz DDR3 RAM, macOS Mojave version 10.14 using Python3 v3.6.3 with Seaborn v0.8.1, Pandas v0.23.4 and NumPy v1.13.3.

3 Results

3.1 Optimization

In practice, the new implementation of the algorithm got a boost in speed with minimal impact on cluster counting accuracy compared with the previous methods (Tables 1 and 2).

Table 1. Comparison of run time of two different algorithms.

Data normalization	Method	Run time (sec)			
		Minimum	Mean	Maximum	Standard deviation
No	*m* sampling	0.039614	4.872610	15.976803	4.918828
No	Geometric decrement	0.045037	31.377525	131.773615	38.138111
Yes	*m* sampling	0.041973	5.316945	21.418652	6.011148
Yes	Geometric decrement	0.040350	30.805692	126.715257	36.140838

Table 2. Comparison of difference from ground truth of two different algorithms.

Data normalization	Method	Absolute difference with ground truth			
		Minimum	Mean	Maximum	Standard deviation
No	*m* sampling	0	0.588235	7	1.219450
No	Geometric decrement	0	0.568627	7	1.220736
Yes	*m* sampling	0	0.823529	7	1.306995
Yes	Geometric decrement	0	0.784314	7	1.285515

On average, the run time is cut down to one-sixth of the original.

The average difference of cluster count does not suffer any visible impact from the change of the implementation.

3.2 Generalization to Higher Dimension

The algorithm performed considerably worse for higher dimension (Table 3).

Table 3. Comparison of performance between data in 2-dimensions and higher dimensions.

Dimension	Absolute difference with ground truth			
	Minimum	Mean	Maximum	Standard deviation
2	0	0.494186	5	0.888504
More than 2	0	1.750000	7	2.140244

The values of *m* in higher dimension ranges from 2 to 3, which is way smaller than that found in 2D data (Table 4).

Table 4. Comparison of final m value between data in 2-dimension and higher dimensions.

Dimension	Final *m*			
	Minimum	Mean	Maximum	Standard deviation
2	5	44.616279	130	36.853168
More than 2	2	2.062500	3	0.245935

It is tempting to explain the inaccuracy by the curse of dimensionality. However, upon further inspection of the datasets, datasets in higher dimensions also happen to have fewer data points compared to 2D datasets (Table 5).

Table 5. Comparison of data count between data in 2-dimension and higher dimensions.

Dimension	Data count			
	Minimum	Mean	Maximum	Standard deviation
2	1481	3425.333333	7500	1856.009912
More than 2	147	810.750000	1024	366.817941

4 Discussion

4.1 Strength

Speed without Sacrificing Accuracy. With all other conditions under control, the implementation of the new algorithm is six times faster without suffering much with regard to accuracy. The final decrement step to find the optimal *m* value also guarantees that the ideal solution is not skipped over by the approach.

Minimal Impact on 2-D Dataset. The performance is on par with previous generations on 2-D datasets.

4.2 Limitations

Sensitive to Local Density Variation within Cluster. Since this algorithm does not fundamentally changes the way it estimates cluster count, it retains a problem that is present in the previous version. It is sensitive to local density which leads to over estimating cluster count in some cases, especially for 2D data.

Inconsistency Between Normalized and Unnormalized Data. In an attempt to unify the behavior of the algorithm across datasets with different ranges, normalization (i.e. conversion to standardized z-score) is applied to all datasets before cluster count estimation. Interestingly, the resulting cluster count estimation differs between normalized and unnormalized data.

Inspection of the detailed logs recording different variables in all loops revealed that the algorithm terminates at different m value for normalised and unnormalised data. Consequently, the estimation of cluster count can differ by as much as 3.

It is possible that local maxima detection is very sensitive to local fluctuations in data density, amplified by normalization. Another possible explanation may be due to the change in data range that affects how the data are divided up and that leads to discrepancy in how the hypercubes are generated in these two versions of the dataset. More work is required to unify the performance between normalized and unnormalized data.

Degeneration of m Value in Datasets with Low Data Amount. We also observe that the final ideal m value almost always degenerates to 2 or 3 for data in higher dimension. This looks like a problem due to dilation of space in higher dimensional space. However, the same issue occurs in datasets at dimension as low as 3. Upon inspection, those datasets share a common characteristic of having a low data count (in the range of hundreds).

Given a very small tolerance, the number of boxes with single data point that terminated the loop for finding the ideal m value will become 0, and therefore pushes the m value to a meaninglessly small value. For example, for a dataset with only 100 observations, the tolerance level of 0.004 means that the algorithm will terminate when the number of single data hypercube drops below $100 \times 0.004 = 0.4$. Since the lowest possible value of m is set to be 2, this guarantees the algorithm to ends with $m = 2$ and therefore unable to accurately estimate the cluster count.

Setting the tolerance at a higher value enables a more accurate cluster count estimation for datasets with low data points, but leads to gross overestimation for those with large data volume. It seems that a dynamic tolerance level or an entirely new termination condition is required for this algorithm to perform uniformly well across datasets with different volume.

4.3 Curse of Dimensionality

In 2D, neighboring grid boxes can be easily identified by the index coordinates of the grid box. However, the same definition of "neighbors" cannot be applied in higher dimension as the number of neighbors increases with the dimension. It does not only pose performance issues for the algorithm, it also makes every hypercube a neighbor to all other hypercubes.

For now, a limit is posed on the number of nearest neighbor in the determination of local density maxima. The limit is arbitrary but well justified as a maximum among the closest few already fulfils the definition of a *local* maxima. Hypercubes of greater "distance" can be omitted.

With a higher tolerance level (t value), datasets with low data volume at higher dimension produces reasonable results. This suggests that this approach may be able to work around the curse of dimensionality that stumps many researches. It is possible that the division of higher dimensional space into hypercubes by our method ensures that the well-ordered nature of our conventional notion of distance (i.e. Euclidean

distance) is preserved as the division is done uniformly and is sensitive to relative distance among data points in each cluster (Table 6).

Table 6. Examples demonstrating the influence of tolerance level on the final *m* value and the resultant cluster estimation count. Configurations that produce the correct result are bolded.

Dataset	Data count	Dimension	Data normalization	Tolerance	Final *m*	Cluster count estimation
Iris species	1481	4	Yes	0.025	2	1
				0.05	4	2
				0.125	**6**	**3**
				0.15	**7**	**3**
				0.225	9	8
Seed dataset	210	7	Yes	0.125	2	2
				0.15	3	5
				0.3	**4**	**3**
				0.35	5	5
				0.45	6	9

4.4 Further Research

More research effort is needed to find out the relationship between local data density, hypercube size and total data points to understand the interaction of these factors and their impact on the original algorithm. This is especially important for understanding the impact of data volume to the performance of the algorithm. As suggested by our results, it is best to have a dynamic termination condition than a static one. The algorithm is not at all useful if the termination condition depends on a parameter input by the user as it throws back the problem of cluster count estimation to humans. It seems to be possible to nail down that moving target if we probe deeper into the relationships among different metrics of the dataset.

Smoothing algorithm may alleviate the problem of overestimation due to local density fluctuation. However, most smoothing algorithms are applied into in 1D or 2D data. Extra attention must be paid in searching for one that can be applied in high dimensional space.

It is also valuable to compare the results from this algorithm with human evaluation. Although we rely on machines to perform routine and computational tasks in a more efficient manner, it is questionable whether machine learning algorithms can *always* generate results "superior" to human judgement, or align with human understanding on how information should be organized.

This problem stems from the fact that even among human, there may be disagreement on what constitutes a cluster. The difficulty in devising a simple way to count clusters may suggest that the potential problem lies in the lack of a clear definition of a "cluster". After all, entities as large as galaxies can form a "cluster" and clusters a supercluster, which may look like a single bright dot on the sky if the observer is far enough. One possible solution to this is to search for clusters in all

different levels. This is, however, an entirely different search topic with extreme difficulty that requires intense research effort.

5 Conclusion

Adapting and tweaking quadratic binary search offers a six-fold boost in cluster estimation speed on average. The interaction between data count and tolerance level to terminate the algorithm poses difficulties in giving meaningful results when the data has low data volume, and results suggest that an adaptive tolerance level may yield good results in higher dimensions and may potentially provide a way out of the curse of dimensionality. Deeper relationship between dimensions, local data density variation, data volume and any other alternative termination conditions is needed for the development of the algorithm to move forward.

Acknowledgements. I must restate my deep gratitude towards Mr. Monte Hancock for recruiting me into the Sirius project and how the project turned my life around.

Inexpressible thanks go to each of my team members: Jishnu for efficient coding and good mistake spotters, Markus for meticulous eyes on overall structure and wording of the paper, Alexis for rigorous critique and proofreading, and Suraj for saving the paper when I most needed help on formatting and putting the pieces together into a complete piece. Many thanks to other workers on the Sirius team working on bits and pieces of this paper. This paper will not be complete without all of your help. Big thumbs up for Lesley the EPM (Executive/Epic Project Manager) for amazing coordination.

References

1. Lo, C.C.-W., et al.: Intermediate information grouping in cluster recognition. In: Schmorrow, D.D., Fidopiastis, C.M. (eds.) AC 2018. LNCS (LNAI), vol. 10915, pp. 287–298. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91470-1_24
2. Kumar, P.: Quadratic Search: A new and fast searching algorithm (An extension of classical Binary search strategy). https://pdfs.semanticscholar.org/3d91/97ecfcc1a16254c8667b0cbd35c93e7f9437.pdf. Accessed 1 Feb 2019
3. Aggarwal, C.C., Hinneburg, A., Keim, D.A.: On the surprising behavior of distance metrics in high dimensional space. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.7409&rep=rep1&type=pdf. Accessed 1 Feb 2019
4. Pele, O.: Distance functions: Theory, algorithms and applications. https://pdfs.semanticscholar.org/c656/f090d5710a524ac26ef1b22310e772fa465c.pdf. Accessed 15 Feb 2019
5. Aggarwal, C.C., Yu, P.S.: The IGrid Index: Reversing the dimensionality curse for similarity indexing in high dimensional space. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=0EBCEB48BA3DE80411807AA7DF3C3A60?doi=10.1.1.129.746&rep=rep1&type=pdf. Accessed 1 Feb 2019
6. Clustering basic benchmark. https://cs.joensuu.fi/sipu/datasets/. Accessed 1 Feb 2019
7. Iris Species. https://www.kaggle.com/uciml/iris. Accessed 1 Feb 2019
8. Seeds dataset. https://www.kaggle.com/rwzhang/seeds-dataset. Accessed 1 Feb 2019